Revista de Informática Teórica e Aplicada - RITA - ISSN 2175-2745
Vol. 26, Num. 3 (2019) 13-25

RESEARCH ARTICLE

Vulnerability Identification on GNU/Linux Operating Systems through Case-Based Reasoning

Identificação de Vulnerabilidades em Sistemas Operacionais GNU/Linux Através de Raciocínio Baseado em Casos

Douglas Santos[1]*, Jéferson Campos Nobre[2]

Resumo: Operating system security has been steadily evolving over the years. Several mechanisms, softwares and guides of best practices of configuration have been developed to contribute with the security of such systems. The process that makes an operating system safer by considering the default level obtained at the installation is known as hardening. Experience and technical knowledge are important attributes for the professional performing this process. In this context, automated rule-based tools are often used to assist professionals with little experience in vulnerability identification activities. However, the use of rules establishes a dependency on developers for the development of new rules as well as to keep them updated. Failure to update rules can significantly compromise the integrity of vulnerability identification results. In this paper, the Case-Based Reasoning (CBR) technique is used to improve tools that assist inexperienced professionals in conducting vulnerability identification activities. The purpose of using CBR is to make inexperienced professionals obtain similar results as experienced professionals. In addition, the dependence on rule developers is diminished. A prototype was developed considering the GNU/Linux system in order to carry out an experimental evaluation. This evaluation demonstrated that the application of CBR improves the performance of inexperienced professionals in terms of the number of identified vulnerabilities.
Keywords: Vulnerability Identification — Case-Based Reasoning — Hardening

Resumo: A segurança em sistemas operacionais vem evoluindo constantemente ao passar dos anos. Diversos mecanismos, softwares e guias de melhores práticas de configuração vêm sendo desenvolvidos para contribuir com a segurança de tais sistemas. O processo que torna um sistema operacional mais seguro considerando-se o nível padrão obtido na instalação é conhecido como *hardening*. A experiência e o conhecimento técnico são atributos importantes para o profissional que executa esse processo. Neste contexto, ferramentas automatizadas baseadas em regras são frequentemente utilizadas para auxiliar profissionais com pouca experiência em atividades de identificação de vulnerabilidades. No entanto, a utilização de regras estabelece uma dependência por desenvolvedores para o desenvolvimento de novas regras assim como para mantê-las atualizadas. A falta de atualização de regras pode comprometer significativamente a integridade dos resultados da identificação de vulnerabilidades. No presente artigo, a técnica de Raciocínio Baseado em Casos (*Case-Based Reasoning* - CBR) é utilizada para aprimorar ferramentas que auxiliam profissionais inexperientes na condução de atividades de identificação de vulnerabilidades. O objetivo da utilização de CBR é fazer com que profissionais inexperientes obtenham resultados semelhantes ao de profissionais experientes. Além disso, diminui-se a a dependência por desenvolvedores de regras. Um protótipo foi desenvolvido considerando o sistema GNU/Linux a fim de realizar uma avaliação experimental. Essa avaliação demonstrou que a aplicação de CBR aprimora o desempenho de profissionais inexperientes em termos do número de vulnerabilidades identificadas.
Palavras-Chave: Identificação de vulnerabilidades — Raciocínio Baseado em Casos — Hardening

[1,2] *Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil*
***Corresponding author**: jcnobre@inf.ufrgs.br
DOI: https://doi.org/10.22456/2175-2745.82079 • **Received:** 18/04/2018 • **Accepted:** 27/08/2019

1. Introdução

A Segurança da Informação vem ganhando importância na medida em que os sistemas computadorizados e softwares se tornam cada vez mais difundidos entre usuários domésticos e organizações. A evolução dos mecanismos de segurança nos sistemas operacionais ganhou visibilidade ainda maior a partir das comunidades de software de código aberto, permitindo que mais pessoas possam trabalhar de forma colaborativa com o objetivo de criar melhores softwares, a exemplo do sistema operacional GNU/LINUX [1] que movimenta milhares de desenvolvedores.

O sistema operacional GNU/LINUX vem se tornando cada vez mais relevante para o mercado [2], e esta crescente demanda acarretou em uma preocupação com segurança. A segurança em sistemas operacionais GNU/LINUX vem sendo objeto de estudos e novos mecanismos de segurança surgem a cada dia. Diversos exemplos destes mecanismos podem ser citados, tais como o controle de acesso mandatório [3], o algoritmo de randomização de endereços de memória [4, 5] e o StackShield [6]. Entretanto, é importante que além da utilização de mecanismos de segurança oferecidos por sistemas GNU/LINUX, também sejam aplicados os devidos cuidados com a configuração de serviços e a política de controle de acesso.

As atividades de remoção de pacotes de aplicações que não são mais necessárias, verificação do conjunto de permissões de acesso e a atualização de pacotes de aplicações relevantes, fazem parte de um processo importante para diminuir a superfície de eventuais ataques a um sistema operacional. O processo conhecido como *hardening*, pode ser definido como o conjunto de atividades que tornam um sistema mais seguro que o nível padrão obtido na instalação do mesmo.

Diversas instituições, tais como MITRE [7], *National Institute of Standards and Technology* [8] e o *Forum of Incident Response and Security Teams* [9] vem publicando guias de melhores práticas de configuração segura [10, 11], que podem ser utilizados durante o processo de *hardening*, auxiliando profissionais em sua condução. A qualidade do resultado do processo de hardening pode ser avaliada após a conclusão de suas etapas.

A avaliação do resultado do processo de *hardening* em um sistema GNU/LINUX é normalmente realizada com a utilização de ferramentas automatizadas, as quais atuam na identificação de fragilidades de segurança. Ferramentas atuais como, por exemplo, a Ferret [12], Lynis [13], Tiger [14] e OpenVAS [15] utilizam um sistema de regras pré-definidas para realizar a identificação de fragilidades. A utilização de regras na identificação de vulnerabilidades é sensível a eventuais mudanças no ambiente, de forma a não detectar pequenas variações, afetando assim a detecção de tais vulnerabilidades. Esta abordagem também estabelece a dependência por desenvolvedores e mantenedores das ferramentas, para que novas regras de verificações sejam criadas.

A dependência por desenvolvedores na tarefa de atualização da ferramenta pode tornar sua base de verificações atrasada comparada a velocidade em que novas vulnerabilidades surgem e de mudanças do ambiente em que se insere. Este fator de defasagem compromete a identificação de vulnerabilidades quando conduzida por um profissional com pouca experiência, o que pode levar a falha na qualidade do *hardening*.

O propósito do presente trabalho é descrever um método de identificar vulnerabilidades de segurança em um sistema GNU/LINUX, auxiliando profissionais inexperientes a obterem desempenho próximo ao de um profissional com maior experiência. A eliminação da dependência por desenvolvedores no processo de atualização das regras de verificação e permitir que pequenas variações no ambiente sejam menos determinantes para a detecção de vulnerabilidades, também são relevantes. Para isto, será aplicada a metodologia Raciocínio Baseado em Casos (*Case-Based Reasoning* - CBR) oriunda do campo de Inteligência Artificial (IA) ao criar uma ferramenta de identificação de fragilidades de segurança que atuará como um assistente, computando e apresentando possíveis vulnerabilidades e soluções pertinentes a cada caso.

2. Fundamentação Teórica

Nesta seção, será possível encontrar maiores detalhes sobre os conceitos de vulnerabilidades, *hardening*, e CBR.

2.1 Vulnerabilidades

Uma vulnerabilidade é um meio por onde uma entidade hostil pode violar a segurança de um sistema com sucesso [16]. As consequências da exploração de uma vulnerabilidade por uma entidade hostil são a perda de informação e a perda de utilidade do próprio sistema [17]. De forma mais abrangente, a exploração de uma vulnerabilidade pode acarretar em impacto na confidencialidade, integridade e/ou disponibilidade de um sistema e informações nele contidas. Algumas iniciativas catalogam vulnerabilidades a fim de facilitar a troca de informações entre interessados, como por exemplo, o projeto *Common Vulnerabilities and Exposure* (CVE), o qual é mantido pela MITRE [18].

Utilizando-se das classes propostas por [17], uma vulnerabilidade pode ser enquadrada como uma falha de design, uma falha de ambiente, uma falha de desenvolvimento ou uma falha de configuração.

A utilização do serviço *snmpd* sob comunidade padrão, pode ser compreendida como uma vulnerabilidade pertencente à classe de falha de configuração. O serviço *snmpd* permite o acesso a informações pertinentes à administração de recursos de um dispositivo de rede. Seu meio de acesso aos recursos se dá através de comunidades, que funcionam como uma senha [19]. A utilização da comunidade padrão, pode acarretar no acesso indevido às informações do dispositivo por uma entidade hostil. Para a classe de vulnerabilidades de falha de design, diversas técnicas de exploração podem ser encontradas [20]. Em alguns casos, pesquisadores publicam programas conhecidos como *exploits*, que são responsáveis por manipular o comportamento do software afetado obtendo alguma vantagem [20].

A exploração bem sucedida de softwares que oferecem serviços de rede podem acarretar em acesso indevido a uma entidade hostil [21]. Este tipo de exploração ocorre quando a entidade hostil não tem acesso a um terminal no sistema ou autorização para tal, apenas o alcance de seus serviços através da comunicação de rede e com pouco ou nenhum privilégio. A vulnerabilidade conhecida como HeartBleed [22] descoberta recentemente, que atinge o software OpenSSL, é um exemplo de vulnerabilidade que pode ser explorada através da rede.

A exploração local, também conhecida como escalada de privilégio [23] se dá através da utilização de um acesso autorizado, ou seja, um usuário autenticado explorando uma vulnerabilidade para obter acesso irrestrito ao sistema GNU/LINUX. Um exemplo de vulnerabilidade que permite esta exploração, está na configuração insegura do recurso *sudo*, que permite que um usuário não privilegiado execute comandos restritos de forma privilegiada.

A exploração de vulnerabilidades representa um risco aos sistemas e informações confidenciais de uma organização. Mensurar o impacto da exploração bem sucedida por uma entidade hostil é uma tarefa complexa. A entidade *International Organization for Standardization* (ISO) define que a probabilidade de uma ameaça explorar uma vulnerabilidade acarreta em um risco e por tanto, deve ser tratado de forma adequada [24]. A devida mensuração do risco, impacto e probabilidades de que vulnerabilidades venham a ser exploradas, pode ser realizada através de um processo de gestão de riscos.

2.2 *Hardening*

Segundo [25], o processo de tornar o sistema operacional mais seguro do que o nível padrão de sua instalação, é conhecido como *hardening*. O conceito de *hardening* também foi adotado por autores para descrever métodos de aprimorar a segurança em softwares [26] e na camada de rede de uma organização [27]. Desta forma, o termo *hardening*, é utilizado para referenciar a melhoria em segurança de algum sistema, podendo ser um software, uma rede e seus dispositivos, assim como um sistema operacional.

Alguns autores contribuem com o tema através de guias de melhores práticas, os quais descrevem atividades para melhorar aspectos de segurança dos sistemas operacionais GNU/LINUX. O guia de configuração segura para o sistema Red Hat Enterprise Linux [28] e o guia de segurança para distribuições GNU/LINUX [11] são exemplos destas contribuições.

O processo de *hardening* em um sistema GNU/LINUX pode ser dividido em três etapas, planejar, configurar e manter [29]. A primeira etapa se caracteriza pela análise e planejamento dos requisitos e funcionalidades que o sistema deverá ter.

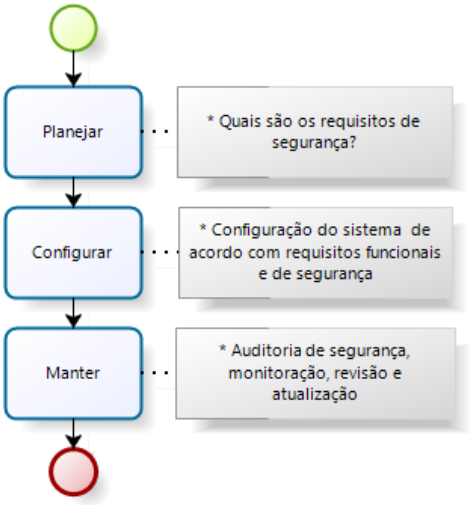
Por exemplo, em uma rede hipotética onde o servidor de rede será responsável por fornecer o horário sincronizado para os demais servidores, o serviço utilizado para este fim pode ser o *NTPD*, que implementa o protocolo Network Time Protocol [30]. Neste caso, deverão ser levantados os requisitos para o pleno funcionamento do serviço *NTPD*. Outro exemplo de atividade da primeira etapa, está na especificação do tipo de sistema de arquivos, como serão configurados os pontos de montagem das partições de disco.

Durante a segunda etapa do processo de *hardening*, a configuração segura de softwares e sistemas operacionais devem receber atenção. Aplicar configurações enxutas e atender eventuais requisitos de segurança são o objetivo da etapa de configuração, desta forma, é possível diminuir a superfície de ataques eliminando possíveis pontos vulneráveis que podem ser explorados para comprometer o sistema.

Ainda durante a etapa de configuração do processo de *hardening*, o administrador deve remover pacotes e serviços desnecessários, alinhar a política de controle de acesso aos recursos do sistema, configurar os serviços remanescentes de forma segura, instalar rotinas de coletas de logs para futuras auditorias e opcionalmente, aderir a alguma tecnologia de detecção ou prevenção de intrusão [11].

Realizar as modificações no sistema operacional de forma a torná-lo mais seguro é importante, porém, manter o sistema seguro é igualmente importante. O objetivo da terceira etapa do processo de *hardening* tem o objetivo de manter o sistema funcionando em harmonia com os requisitos de segurança. Nesta etapa do processo de *hardening*, deve-se estabelecer e respeitar uma política de atualização de softwares e do sistema operacional. Também é importante revisar periodicamente logs para detectar possíveis anomalias, realizar verificações rotineiras para identificar vulnerabilidades, estabelecer e respeitar uma política de backup, assim como avaliar sua eficácia com regularidade [11]. A figura 1 contextualiza algumas das atividades presentes em cada etapa do processo de *hardening*.

Figura 1. Etapas de um hardening e exemplos de atividades



Fonte: Elaborado pelo Autor.

Na etapa de planejamento, perguntas elementares devem ser respondidas para dar suporte às atividades das etapas conseguintes. A etapa configurar compreende as atividades perti-

nentes para aumentar o nível de segurança do sistema operacional e seus softwares. Durante a etapa manter, atividades de preservação da segurança do ambiente são conduzidas.

O conhecimento do profissional que conduz as atividades do processo de *hardening* determinam a qualidade do resultado final. Para que um maior número de profissionais consigam conduzir estas atividades, os guias de referência [11, 28] podem ser consultados.

2.3 Raciocínio Baseado em Casos (*Case-Based Reasoning* - CBR)

Segundo [31], a IA é a área de conhecimento que se propõe a estudar a automação de comportamento inteligente. Existem quatro classificações principais que tentam justificar uma definição de acordo com o comportamento de um algoritmo que utiliza a Inteligência Artificial (IA). Sistemas que pensam como humanos, sistemas que pensam racionalmente, sistemas que agem como humanos e sistemas que agem racionalmente, são vistos como os principais vértices da Inteligência Artificial (IA) [32].

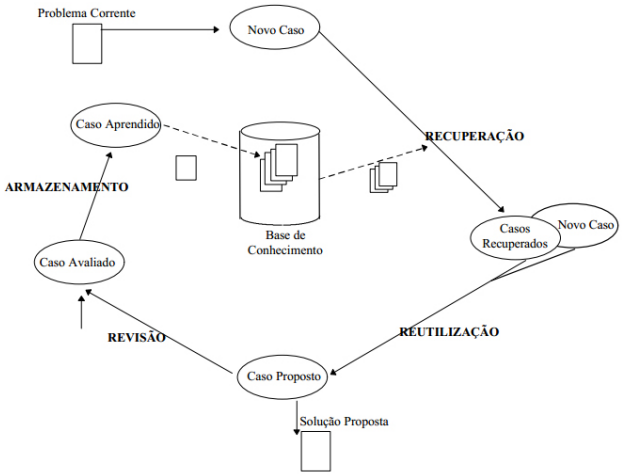
Existem vários tipos de abordagem de sistemas inteligentes, neste trabalho foi abordada metodologia CBR, dada sua capacidade de solucionar problemas de variados níveis de complexidade e por sua abordagem objetiva e simples, tornando-se viável sua aplicação para a solução da proposta deste trabalho.

Sistemas especialistas são modelos de algoritmos de IA destinados a raciocinar sobre algum campo de conhecimento específico, com o objetivo de resolver problemas [33]. Uma abordagem de sistema especialista baseada na metodologia CBR busca solucionar problemas baseando-se em experiências passadas [34]. Isto significa que para que seja possível chegar a uma solução, o algoritmo necessita reconhecer o problema descrito com base em experiências vividas e aplicar a mesma solução ou adaptá-la para a situação atual. Sistemas CBR podem apoiar diversas áreas, e por esta flexibilidade são encontrados em sistemas industriais [35], na área da saúde [36], área jurídica [37], e em gerenciamento de redes [38].

Ao adaptar a solução aplicada a um caso antigo para resolver um novo problema, um sistema CBR aproxima-se do raciocínio humano cognitivo. Constituído por um ciclo de quatro etapas, um sistema CBR deve compilar uma base de casos estruturada, a fim de recuperar informações rapidamente quando necessário [39]. O ciclo é composto pelas etapas de recuperação, reutilização, revisão e armazenamento conforme ilustra a Figura 2.

Durante a primeira etapa, algoritmos de similaridade são aplicados para identificar semelhanças entre o caso armazenado e o problema atual, para que somente as soluções mais congruentes com o problema sejam recuperadas. A segunda etapa compreende atividades nas quais a solução é aplicada e se necessário adaptada para se enquadrar no atual problema. Na terceira etapa, medidas de revisão da solução são aplicadas para avaliar seu resultado. A quarta e última etapa se caracteriza pelo armazenamento da solução aplicada ao problema

Figura 2. Etapas da metodologia CBR



Fonte: Adaptado de [38].

atual, se esta foi adaptada e gerou uma nova experiência, tornando o sistema mais inteligente na medida em que novos casos são aprendidos.

A primeira etapa se caracteriza por influenciar diretamente no desempenho de um sistema CBR. Para que o algoritmo tenha um desempenho razoável, cada caso em sua base de informações deve ser descrito com o maior número de detalhes possível, do contrário pode ocorrer a reutilização de casos ineficientes para a resolução de problemas. Algumas considerações devem ser feitas também em relação ao mecanismo de escolha de casos aplicáveis para o problema de momento. O mecanismo de identificação de casos correspondentes deve ser alimentado pela descrição do problema atual e de problemas armazenados, portanto devem haver campos com nomes iguais no modelo de descrição de casos [38].

2.3.1 Representação e Banco de Casos

Segundo [40], um caso corresponde a um fragmento de conhecimento representando uma experiência que ensina uma lição para atingir determinado objetivo. A descrição de um caso reúne elementos fundamentais que caracterizam um problema e as informações que levem à sua solução. Uma das abordagens mais simples de se representar um caso é através de pares de atributo-valor organizadas em um vetor [34, p. 66, 67]. Essa abordagem permite armazenar dados numéricos, textuais, e símbolos [34, p. 166].

Um exemplo de representação de pares de atributo-valor pode ser compreendido à partir da descrição de um processo em um sistema GNU/LINUX, a partir de seus atributos, tais como, nome do sistema operacional, nome do processo e dono do processo, conforme relaciona a Tabela 1. Desta forma, pode-se assumir que caso é um conjunto finito de pares de atributo-valor que visam identificar um problema e sua respectiva solução [34, p. 67].

A metodologia CBR busca reproduzir o raciocínio humano ao utilizar a memória aplicada a solução de problemas

Tabela 1. Exemplo de representação de pares de atributo-valor

Atributo	Valor
Sistema Operacional	GNU/LINUX
Nome do Processo	Apache2
Dono do Processo	www-data

Fonte: Elaborada pelo Autor.

conhecidos. Desta forma, uma estrutura de armazenamento de casos se faz necessária. Uma pessoa busca resolver problemas em seu cotidiano, entre outras formas, baseada em situações vividas anteriormente. Para isso, uma pessoa utiliza sua memória para relembrar estas situações. De fato, o evento de buscar e encontrar um caso pertinente na base, é conhecido como "lembrar"na metodologia CBR [41].

Para que os casos sejam armazenados de forma a viabilizar atividades de busca, a organização da base de casos pode se dar de três formas diferentes: estruturada, não-estruturada e *flat* [34, p. 67]. O formato estruturado permite que sejam criadas categorias e assim formar hierarquias para tornar o processo de busca mais eficiente. A forma não-estruturada pode ser aplicada em conjunto a textos ou imagens. No formato *flat*, os casos são armazenados de forma linear sem qualquer hierarquia [42, 35]. Em uma base de casos organizada no formato *flat*, os casos são recuperados através de pesquisas em todos os dados existentes na base [42].

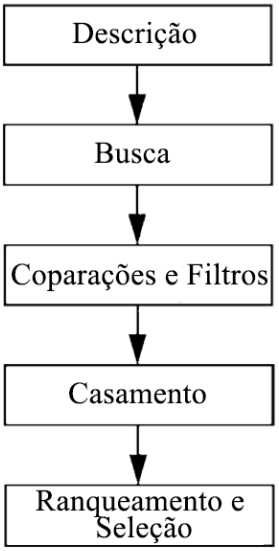
2.3.2 Recuperação de Casos

Um sistema CBR aplica soluções aprendidas em experiências passadas para resolver um problema atual. Para que a melhor solução seja encontrada, um processo de recuperação de casos deve ser acionado. O processo de recuperação deve pesquisar na base de casos utilizando algoritmos específicos de busca, para que os casos que melhor se enquadram na descrição do problema atual sejam selecionados [43].

O fluxo exibido na Figura 3 mostra os passos da etapa de recuperação, onde a descrição do problema é recebida para então realizar a busca. A busca é realizada de acordo com o modelo escolhido, e em uma estrutura *flat* por exemplo, a busca pode ser sequencial. Isto significa que todos os casos serão analisados como potencial solução para o problema atual. Adicionalmente, filtros podem ser aplicados em situações onde existe uma hierarquia na estrutura dos casos [38]. Mecanismos de similaridade são aplicados para encontrar casos semelhantes ao problema atual. Ao encontrar casos semelhantes, índices de similaridade são atribuídos para serem utilizados na ordenação dos casos recuperados. Através da ordenação dos casos, é possível apresentar casos mais similares e que portanto a solução tenha maior potencial de solucionar o problema atual [43].

A eficiência da etapa de recuperação depende da forma adotada para a representação dos casos, da estrutura de armazenamento, e da forma de medir a similaridade entre os casos [34, p. 272]. Por exemplo, em uma base de casos or-

Figura 3. Fluxo do processo de recuperação de casos



Fonte: Adaptado de [43].

ganizada no formato *flat*, os casos são recuperados através de uma pesquisa sequencial completa, avaliando cada caso existente [42]. Este fator impacta diretamente na eficiência do mecanismo de recuperação, uma vez que todos os casos serão acessados, mesmo aqueles que acabam por ser irrelevantes. Diferentemente de uma organização hierárquica, onde grupos são formados e a busca pode ocorrer em apenas um grupo, fracionando o esforço de pesquisa e portanto acelerando o processo. A recuperação sequencial é de fato a mais simples e encontrada em sistemas onde a base de dados é pequena ou onde os casos possuem poucos pares de atributo-valor [34, p. 275].

Existem diversos algoritmos utilizados para a fase de recuperação de casos. Entre tais algoritmos, o mais comum é o de vizinhança (*Nearest-Neighbor Matching*) [44]. No algoritmo de vizinhança, o objetivo é encontrar os casos mais próximos ou ainda, mais semelhantes ao problema proposto. Na recuperação de casos através do algoritmo de vizinhança, o caso é recuperado quando a soma ponderada de seus atributos é maior que a de outros casos na base [42]. Para isto, uma função de similaridade é aplicada no comparativo entre o dado de cada atributo de um caso, com o dado do respectivo atributo no problema proposto [44].

A análise de similaridade é vista por alguns autores como a principal etapa para um sistema CBR [45] pois as etapas subsequentes dependem dela. Visto que a solução para um problema é fundamentada em experiências passadas [38, 41], é preciso que o sistema seja capaz de analisar as semelhanças entre situações vividas com o problema atual, para então selecionar as possíveis soluções. Quando o algoritmo CBR encontra em sua base de casos um caso semelhante ao problema proposto, este evento é chamado de lembrança [41]. Publicações anteriores [45, 46, 38], mostram que existem va-

riadas formas para se medir a similaridade entre casos da base de casos e o problema proposto.

Alguns dos algoritmos que podem ser utilizados para o papel de função de similaridade são a distância Euclideana [45], a função de Levenshtein [45], a similaridade de cosseno [47] e a baseada em compressão [45]. Estes algoritmos são compatíveis com o formato de pares de atributo-valor utilizado na representação de casos, deste modo, a função de similaridade pode ser aplicada para identificar se os atributos de ambos são semelhantes ou não [46, 38]. Em uma análise de similaridade, o valor retornado pela função escolhida, varia entre o intervalo de números reais 0 e 1, onde 0 indica que os dados são completamente diferentes e 1 representa semelhança total [45, 46].

Ao utilizar o sistema de soma ponderada sugerido pelo algoritmo de vizinhança para realizar a recuperação, um atributo pode receber maior relevância que outro e portanto, um peso maior na análise de similaridade [38]. Em suma, o cálculo da semelhança entre os casos deve aplicar uma função de similaridade para cada atributo compatível entre o problema atual e o caso recuperado, multiplicando o resultado desta função pelo respectivo peso do atributo. Este algoritmo pode ser representado através da Equação 4.

Figura 4. Equação de Similaridade

$$Similaridade(T,S) = \sum_{i=1}^{n} f(T_i,S_i) \times w_i \quad (1)$$

Fonte: Adaptado de [44].

Na equação 4, S representa o caso de origem (presente na base de casos), T representa o caso alvo (o problema proposto), n representa o número total de atributos em cada caso, f representa a função de similaridade aplicada, i representa cada atributo individualmente, e w representa o peso para o atributo i.

É importante notar que:

- Não há obrigatoriedade de correspondência de atributos no algoritmo de vizinhança, ou seja, um atributo pode existir em um caso da base de casos, mas não existir no problema proposto. Neste caso, o atributo sem correspondente deve ser ignorado [38]. Também será ignorado aquele atributo que possuir seu peso igual a 0 (zero) de acordo com a equação descrita;

- A função de similaridade exerce um papel fundamental. É o componente que determinará o quão parecidos são dados dos atributos de um caso e os do problema proposto. Esta função pode ser baseada em diversos algoritmos diferentes, como a função de Levenshtein e a distância Euclideana. Os algoritmos utilizados devem ser escolhidos de acordo com o tipo de dado a ser analisado. Alguns algoritmos são mais indicados para valores numéricos e outros para textos [45].

- Em um sistema CBR, o caso que obtiver maior grau de similaridade não necessariamente fornecerá a melhor solução para o problema. Neste caso, uma lista de casos ordenada de forma decrescente em relação à similaridade com o problema deve ser fornecida, permitindo que a melhor solução seja selecionada [38, 35].

O comparativo entre atributos deve utilizar o modelo que melhor se enquadra no tipo de dado em questão ou nos requisitos do tipo de atributo. O modelo absoluto, por exemplo [46], pode ser utilizado quando precisamos saber apenas se o valor de um atributo é igual ou não ao valor de outro. Já no modelo relativo [46], existe a possibilidade de obter um índice de semelhança entre dois dados. Isto significa que é possível conhecer o grau de semelhança entre duas frases, como por exemplo, determinar que as frases as frases "A casa é azul"e "A casa é verde", possuem um grau de semelhança de 69% ao aplicar o algoritmo de Levenshtein como função de similaridade.

No modelo relativo, quando os dados dos atributos correspondentes são iguais, a função de similaridade deve retornar o valor 1 (um), quando são completamente diferentes, deve retornar 0 (zero), e quando os dados possuem certo grau de semelhança, deve retornar valores reais pertencentes ao intervalo [0,1], ou seja, valores intermediários na proporção de semelhança entre os dados [38, 46].

Após aplicar a função de similaridade e a equação completa proposta para o algoritmo de vizinhança, obtém-se o índice de similaridade do caso em relação ao problema atual. Este valor pode ser utilizado na ordenação dos casos para que os casos com maior probabilidade de fornecerem uma solução aplicável, sejam fornecidos prioritariamente. Este processo conclui a etapa de recuperação de casos, onde seu resultado será uma lista de casos ordenada de acordo com seu índice de similaridade em relação ao problema proposto [35].

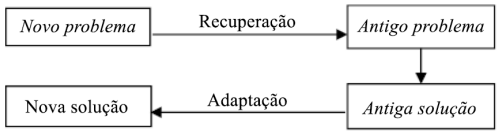
2.3.3 Reutilização e Adaptação

Após o mecanismo de recuperação fornecer casos com grande potencial de solucionar o problema atual, pode ocorrer situações onde a proposta de solução carece de algum detalhe para efetivamente resolver o problema, nestes casos pode-se optar por adaptá-la. Quando isto ocorre, o sistema deve armazenar este novo conjunto de informações como um novo caso [38]. Por outro lado, caso a solução proposta se enquadre perfeitamente na situação, de forma que não seja necessário adaptá-la, pode-se então aplicá-la sem alterações. O processo de aplicação da solução sem adaptação é conhecido como *adaptação nula* [48].

A adaptação nula pode ocorrer em situações onde a solução é genérica ou muito simples, como por exemplo, quando a solução aponta que determinado software deve ser atualizado para a sua última versão para corrigir uma vulnerabilidade específica. Porém, quando alguma modificação na solução proposta se faz necessária, o usuário pode conjecturar sobre quais aspectos devem ser alterados para contemplar os detalhes ausentes. Ao final deste processo, as alterações realizadas

pelo usuário devem compôr um novo caso conforme descrito em seções anteriores. Este cenário de adaptação manual de soluções é conhecido como adaptação baseada em crítica [49]. A Figura 5 ilustra o processo de adaptação de uma solução após a recuperação de um caso.

Figura 5. Fluxo de adaptação da solução de um caso



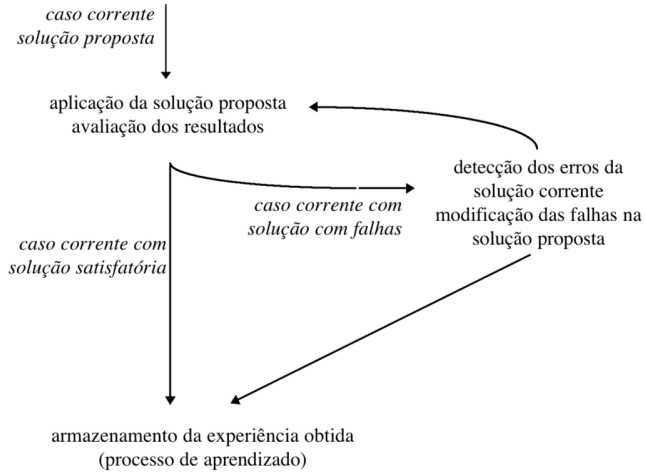
Fonte: Adaptado de [34, p. 74].

Conforme a Figura 5, o problema atual é comparado com antigos problemas para obter uma possível solução conhecida, onde a adaptação desta antiga solução leva a uma nova solução, portanto, uma nova experiência.

2.3.4 Revisão e Armazenamento

A capacidade de aprendizado de um sistema que utiliza a metodologia CBR está diretamente relacionada à identificação e armazenamento de uma nova experiência [38]. Desta forma, em situações onde a solução para o problema passa por adaptação, se faz necessária a avaliação da eficácia da nova solução. Se esta nova solução não obter resultado satisfatório, ela deve ser então reparada e uma nova solução deve ser formulada. A solução deve ser armazenada somente quando for satisfatória [38], conforme pode ser visto na Figura 6.

Figura 6. Fluxo de revisão e reparo de uma solução



Fonte: Adaptado de [38].

O sistema CBR pode aplicar automaticamente a solução que melhor se enquadra no problema proposto, ou deixar a decisão final para o usuário [34, p. 74]. Em ambas as situações a etapa de revisão requer avaliação da solução por parte de um usuário, o que significa que o processo de avaliação da solução aplicada deve ser realizada fora do sistema CBR [50]. Isto ocorre pois dependendo da aplicação do sistema CBR a solução pode levar algum tempo para ser percebida, como por exemplo, em um sistema de apoio a tratamento médico, pode ser necessário aguardar resultados de exames completos de um paciente para que o tratamento recomendado tenha sua eficácia confirmada.

3. Materiais e Métodos

Reconhecer padrões que podem levar à identificação de uma vulnerabilidade passiva de ser explorada, é uma habilidade importante para um profissional que atua em um processo de *hardening*, teste de intrusão [51] ou análise de vulnerabilidades. Esta habilidade é adquirida conforme a evolução da experiência deste profissional. Naturalmente, um profissional com este perfil tem custo elevado em comparação a outro profissional de mesma função e com menos experiência. Também é verdade que profissionais com menos experiência recorrem a ferramentas automatizadas para auxiliar na identificação de vulnerabilidades, fato que nem sempre acarreta em um bom resultado final, devido a limitações das ferramentas atuais.

Em alguns casos, profissionais mais experientes tentam criar manuais de como executar as tarefas de identificação de vulnerabilidades com o intuito de fomentar o conhecimento dos menos experientes, pois quanto menor for o tempo de treinamento do profissional inexperiente, mais rápido se dá o retorno do investimento realizado ao contratá-lo. A criação destes manuais pode ser entendido como uma forma de sintetizar o conhecimento obtido através da experiência do profissional.

Este trabalho propõe a utilização da metodologia CBR para o processo de identificação de vulnerabilidades em um sistema GNU/LINUX. O objetivo é de sintetizar o conhecimento de profissionais experientes com o intuito de auxiliar profissionais pouco experientes em atividades de identificação de vulnerabilidades em sistemas GNU/LINUX. A aplicação da metodologia CBR permite flexibilidade na criação de casos, eliminando a dependência por desenvolvedores na atividade de criar novos casos de vulnerabilidades e também permite o aprendizado constante de acordo com a utilização do sistema e com a aplicação das soluções por ele propostas.

A metodologia CBR é composta por quatro etapas, recuperação, reutilização, revisão e armazenamento, conforme descrito na seção 2.3 . Trabalhos anteriores [38, 35, 34, 45, 46], mostram que existem diversas formas de se aplicar cada uma destas etapas. Neste trabalho foram utilizados as funções de similaridade baseadas em cosseno e Levenshtein pois ambas apresentaram resultados satisfatórios nos testes realizadas. Para o algoritmo de recuperação foi utilizado o *Nearest-Neighbor Matching*, ou algoritmo de vizinhança por sua vasta documentação, simplicidade, eficiência e ser comum entre as aplicações CBR, conforme mencionado anteriormente. Para a representação de casos compatível com o algoritmo de vizinhança, foi utilizado o modelo de pares de atributo-valor. Já para o processo de armazenamento dos casos e indexação da base de casos, como prova de conceito, foi utilizada a estrutura flat, por oferecer simplicidade ao mecanismo de recuperação e também a fácil implementação, embora não

seja a melhor alternativa quanto à eficiência. Desta forma, os algoritmos e modelos mais comuns na metodologia CBR foram utilizados, evidenciando assim a aplicabilidade desta metodologia para as atividades de criação de casos de vulnerabilidades e suas correspondentes identificações em sistemas GNU/LINUX.

3.1 Implementação do Protótipo

Para que fosse possível avaliar a viabilidade da aplicação da metodologia CBR para o propósito deste trabalho, tornou-se necessário desenvolver um protótipo funcional. O protótipo, chamado de V-Gather, possui a capacidade de cadastrar novos casos, identificar vulnerabilidades a partir da recuperação de casos e aplicação de seus algoritmos de similaridade, sugerir soluções, permitir avaliação da solução proposta, a adaptação da solução em situações onde existe a necessidade de complemento da mesma e o aprendizado de novas experiências, de acordo com a metodologia CBR.

A aplicação da metodologia CBR através de um protótipo, terá o papel de atuar como um sistema de recomendação, ou seja, sugerir possíveis vulnerabilidades e suas respectivas soluções, para que o profissional com pouca experiência consiga desempenhar o papel com desempenho mais próximo ao de um especialista. Este processo deve ser realizado sem a necessidade de escrita de códigos de linguagens de programação. Isto significa, que pessoas sem o conhecimento de programação, devem ter capacidade de definir ou aprender novos casos de vulnerabilidades e suas respectivas soluções.

No protótipo deste trabalho foram utilizados os métodos organizacionais de pares de atributo-valor e estrutura *flat* em conjunto com um sistema gerenciador de banco de dados. Conjunto este que possui as vantagens de fácil compreensão, integraçào com sistemas de banco de dados relacionais e também o suporte a grandes bases de conhecimento caso a representação do caso possua poucos pares de atributo-valor [34, p. 168]. A desvantagem da adoção do armazenamento de casos baseado em *flat*, está na eficiência na recuperação dos casos, uma vez que todos os casos existentes na base precisam ser acessados.

Durante o desenvolvimento do protótipo resultante deste trabalho, os mecanismos de busca sequenciais em conjunto com o algoritmo de vizinhança foram utilizados na fase de recuperação de casos. Em conjunto com o algoritmo de vizinhança, foi utilizado como função de similaridade o algoritmo de Levenshtein. Entretanto, após alguns testes foi possível notar que em casos onde o dado analisado era numérico, a conotação textual para calcular seu coeficiente de similaridade foi utilizada. Isto significa, que o coeficiente de similaridade obtido ao usar algoritmo de Levenshtein o entre o número 75 (setenta e cinco) e 74 (setenta e quatro) foi de 0.5, ou seja, um texto de dois caracteres onde apenas um é diferente. Desta forma, identificou-se a necessidade de uma alternativa com maior eficácia para o cálculo de similaridade entre dados numéricos. A função de similaridade de cosseno apresenta características que se encaixam neste requisito [47] e acabou por ser utilizada. O algoritmo de similaridade de cosseno é aplicado para avaliar a proximidade de dois vetores de dados numéricos.

3.2 Estrutura do Protótipo

O protótipo foi estruturado baseado no conceito gerente-agente [52], onde o módulo agente coleta informações dos sistemas GNU/LINUX monitorados e envia ao módulo gerente, que processa estas informações. Também foi desenvolvido um módulo web, onde o usuário pode visualizar os casos ordenados encontrados. O fluxo dos dados inicia no agente, que coleta informações dos processos em execução e submete ao gerente que tem o papel de estruturar estes dados no formato de atributo-valor, compatibilizando com a metodologia CBR e o algoritmo de recuperação adotado neste trabalho. O gerente também processa as informações dos agentes em busca de casos similares, ordenando o resultado em uma lista.

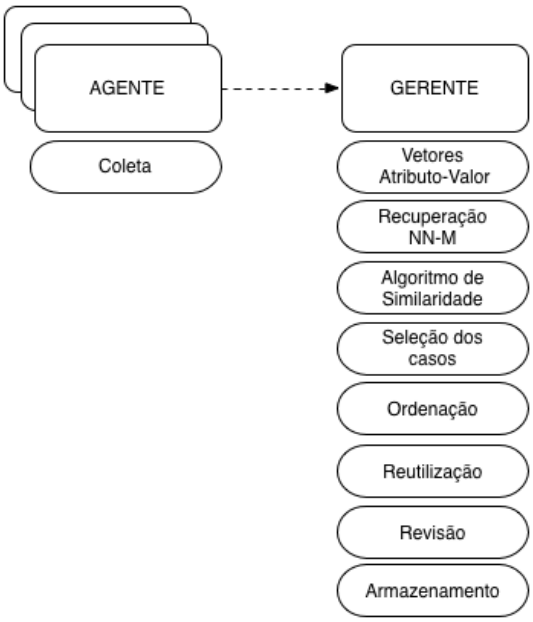
Durante a etapa de recuperação de casos, as informações recebidas pelo agente devem configurar o problema atual ao serem organizadas no formato atributo-valor, desta forma o sistema deve aplicar o algoritmo de busca sequencial na base de casos para localizar os vizinhos mais próximos. A função de similaridade então é aplicada para analisar o grau de semelhança entre dados dos atributos correspondentes entre o problema atual e o caso recuperado. No protótipo deste trabalho, foram utilizados como funções de similaridade os algoritmos de Levenshtein para dados textuais e função cosseno de similaridade para valores numéricos. O resultado da função de similaridade é multiplicado pelo peso do atributo, formando-se assim a equação de similaridade de forma completa. O resultado dos casos mais próximos são então alimentados na base de dados para que possam ser futuramente ordenados e apresentados ao usuário.

O protótipo deve servir como um sistema de recomendações, desta forma, as fases seguintes da metodologia CBR, são conduzidas pelo usuário através do módulo web. Após autenticar-se no painel do módulo web, o usuário tem acesso à lista de todos os casamentos ocorridos entre problema (processos detectados pelo agente) e caso da base, ordenados de forma decrescente em relação ao resultado da etapa de similaridade, conforme descrito na seção de 2.3.2.

Ao selecionar o vizinho que melhor se enquadra no problema, o usuário também estará decidindo qual solução deve ser aplicada. O protótipo deve alterar o estado deste caso, passando a enquadrá-lo na fase de revisão. Isto significa, que o usuário deve executar manualmente as atividades descritas na solução proposta pelo caso, e avaliar a sua eficácia levando em conta as informações descritas no caso ou recebendo o apoio de um especialista. Neste momento, pode-se admitir que o problema e o caso da base, estabeleceram uma relação verdadeira e que o problema pode ser compreendido como uma real vulnerabilidade.

Após a execução das atividades de correção da vulnerabilidade e a revisão da eficácia das medidas corretivas tomadas, o usuário deve fornecer um parecer ao sistema. Através do

Figura 7. Modelo estrutural do protótipo



Fonte: Elaborado pelo Autor.

módulo web, é possível adaptar a solução fornecida para o problema no momento da apresentação de todos os casamentos detectados, e também após a revisão da eficácia das medidas corretivas. Sempre que o usuário decide por adaptar a solução proposta, uma nova experiência é gerada sendo então aprendida pelo protótipo. Nas oportunidades futuras, esta nova experiência será utilizada nos algoritmos de recuperação e similaridade, conforme prevê a metodologia CBR.

3.3 Papéis dos Atores

O conceito de atores [53] pode ser adotado para contextualizar os papéis dos indivíduos que podem interagir com o protótipo. Estes atores podem ser compreendidos como o profissional especialista, e o profissional iniciante. O profissional que detém conhecimentos avançados sobre identificação, exploração e correção de vulnerabilidades em sistemas GNU/LINUX, é visto como especialista. Seu conhecimento a respeito deste assunto, permite que seja ele o responsável por definir novos casos de vulnerabilidades. Também é desejável que o especialista tenha amplo conhecimento sobre os sistemas GNU/LINUX que serão analisados com o auxílio do protótipo deste trabalho.

O conhecimento do ambiente de sistemas GNU/LINUX e do ambiente organizacional é importante para definir o grau de relevância das vulnerabilidades e também contribui na tarefa de definir pesos aos atributos do caso, no momento de sua criação. Por exemplo, quando necessário criar um caso de vulnerabilidade relacionada a um serviço do sistema, os atributos versão e porta TCP, podem receber maior relevância (peso) do que o atributo destinado aos argumentos do processo.

O profissional iniciante é o ator responsável pela execução do módulo agente no sistema GNU/LINUX que terá seus processos analisados em busca de vulnerabilidades. Também é o profissional iniciante o responsável por aplicar ou adaptar a solução proposta para os casos encontrados (vulnerabilidades), avaliar a eficácia das medidas corretivas e fornecer o parecer para que o sistema aprenda com suas experiências. O profissional iniciante não deve utilizar o mecanismo de cadastro de casos, pois esta tarefa requer alto nível de conhecimento de sistemas e da estrutura em que os sistemas se inserem em uma organização.

3.4 Definições do Gerais

Durante a fase de recuperação de casos, a equação de similaridade leva em consideração o peso dos atributos para formular o seu resultado, duas considerações devem ser levadas em conta neste contexto, (*i*) é preciso definir os possíveis pesos permitidos para que o profissional experiente possa utilizar no momento da criação do caso, e (*ii*) também é preciso definir qual será o valor utilizado para decidir se o caso deve ser apontado como uma possível vulnerabilidade, ou não. Isto é, a comparação do problema atual com cada caso da base de casos recebe um valor resultante da equação de similaridade, este valor deve ser comparado com o valor delimitador, onde somente casos que obtiverem similaridade maior que o delimitador, devem ser apresentados ao usuário como uma possível vulnerabilidade. Este procedimento foi adotado para evitar que casos e problemas com baixa similaridade sejam tratados como uma possível solução, funcionando assim como um filtro. Este valor foi obtido de acordo com a observação do comportamento da detecção em vários experimentos, através de avaliação empírica, sendo o valor de número 10 o que apresentou o melhor resultado.

Durante o desenvolvimento do protótipo e a aplicação dos algoritmos envolvidos no raciocínio baseado em caso, foram necessárias algumas definições. A criação de um novo caso, deve ser realizada por um profissional experiente e que detém conhecimento do ambiente organizacional onde o sistema analisado se insere. Ele também será responsável por atribuir pesos para os atributos que compõe o caso, conforme descrito anteriormente nas seções 2.3.2 e 3.3. A escolha dos valores possíveis para o peso dos atributos foi baseada em um estudo anterior [38], e variam entre 1 para o atributo de baixa relevância, 3 para o atributo de média relevância, e 5 para o atributo de alta relevância.

3.5 Avaliação

Durante a fase de experimentação e avaliação da eficácia do protótipo, foi utilizado o o método de delineamento de grupos contrastantes, onde participaram 8 indivíduos no total. Um questionário foi desenvolvido e endossado por três professores da graduação tecnológica em segurança da informação, de uma universidade localizada no município de São Leopoldo-RS, que atuaram como juízes. O objetivo do questionário foi o de identificar diferentes níveis de experiência e conhecimento técnico entre os participantes do experimento.

O questionário foi aplicado aos indivíduos, que cursavam a graduação tecnológica em segurança da informação,

dos 8 selecionados, 50% cursavam entre os primeiros três semestres e o restante cursava entre os três semestres finais. Como experiência no processo de identificação de vulnerabilidades, foi considerado que a graduação tecnológica em segurança da informação oferece as disciplinas de fundamentos de segurança da informação no segundo semestre, segurança de aplicações no quarto semestre, e também a disciplina de testes de invasão de rede no quinto semestre. Entende-se que para os indivíduos cursando semestres finais, o maior conhecimento no contexto da segurança da informação contribua para a atividade de identificação de vulnerabilidades em sistemas GNU/LINUX.

Para avaliar a eficácia da solução proposta por este trabalho, os grupos de indivíduos foram submetidos à atividades de identificação de vulnerabilidades em um sistema GNU/LINUX, onde as atividades foram divididas em três diferentes etapas. A primeira etapa, se reserva a identificar vulnerabilidades sem a utilização de qualquer ferramenta especialista, de forma que somente comandos nativos do sistema operacional estariam ao alcance do participante do experimento. Na segunda etapa, diferentemente da primeira, foi permitido aos indivíduos a utilização das ferramentas Lynis [13], Tiger [14] e OpenVAS [15], que utilizam sistemas de regras para a identificação de vulnerabilidades. A terceira etapa, foi destinada exclusivamente à utilização do protótipo V-Gather, que aplica a metodologia CBR.

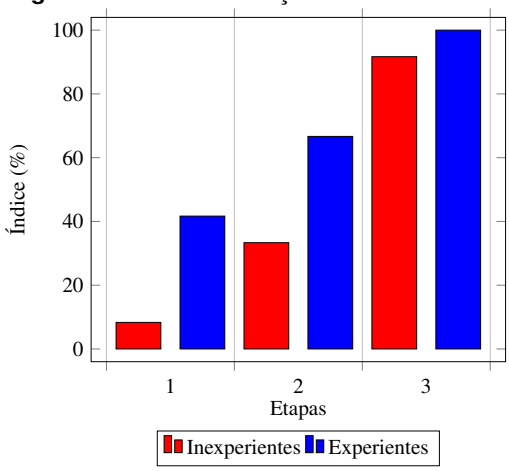
O escopo das atividades de identificação de vulnerabilidades foi definido como os processos em execução no sistema GNU/Liunx, sem haver a necessidade de verificações de credenciais, aplicações web, ou controle de acesso. Três vulnerabilidades foram introduzidas propositalmente. A primeira vulnerabilidade localizada no processo de nome *snmpd*, a segunda no processo *apache2*, e a terceira vulnerabilidade no processo *rmid*. As vulnerabilidades pertencem às classes de falha de configuração e falha de desenvolvimento, de acordo com a taxonomia proposta por [17].

4. Resultados Obtidos

Para analisar os resultados foram aplicados métodos estatísticos, cujo principal objetivo foi o de medir a proximidade dos resultados de cada etapa para os grupos de alunos dos semestres finais em relação ao grupo de alunos dos semestres iniciais. A média de identificação de vulnerabilidades para cada grupo por etapa, foi utilizada como meio de comparação e avaliação do desempenho dos indivíduos. Como forma de facilitar a representação destes dois grupos, os alunos pertencentes aos semestres finais foram incorporados em um grupo chamado de *Experientes*, da mesma forma, os alunos pertencentes aos semestres iniciais foram chamados de *Inexperientes*.

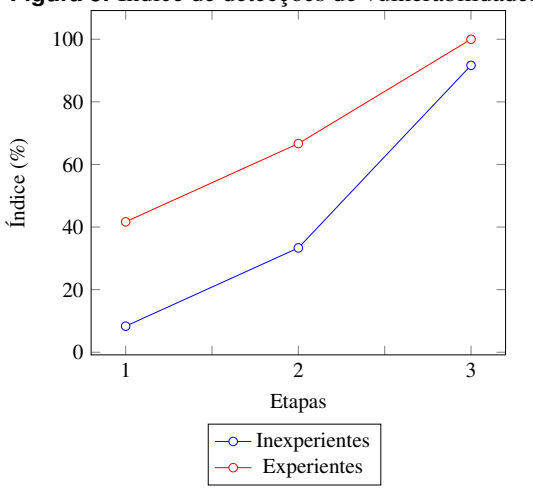
Ao avaliar os resultados obtidos, percebeu-se que para o grupo de indivíduos pertencentes ao grupo Experientes, a média de identificações de vulnerabilidades nas três etapas foi superior. Na primeira etapa a média de identificações das vulnerabilidades foi de 41,67%, na segunda etapa a média foi de 66,67% e na terceira etapa o resultado foi de 100%. Já o grupo de alunos pertencentes ao grupo Inexperientes obtiveram médias 8,33% para a primeira etapa, 33,33% para a segunda etapa e 91,67% para a terceira etapa. As Figuras 8 e 9 ilustram este resultado, onde se torna clara a proximidade dos resultados, principalmente na terceira etapa.

Figura 8. Índice de detecções de vulnerabilidades



Fonte: Elaborado pelo Autor.

Figura 9. Índice de detecções de vulnerabilidades



Fonte: Elaborado pelo Autor.

É possível notar através das Figuras 8 e 9 que o índice de detecção de vulnerabilidades em cada etapa do experimento foi aumentando gradativamente tanto para o grupo Experientes quanto para o grupo Inexperientes. Também é possível notar que na terceira etapa, ao beneficiarem-se da metodologia CBR os resultados obtidos pelo grupo Inexperientes se aproximou demasiadamente dos resultados obtidos pelo grupo Experientes. A Tabela 2 apresenta a relação das médias de identificação de vulnerablidades em cada etapa. O desvio padrão obtido nas etapas 1,2 e 3 para o grupo Inexperientes foi de respectivamente 0.5, 0.82, 0.50, já para o grupo de Experientes o desvio padrão foi de 0.5, 0 e 0 respectivamente.

A medição do desvio-padrão é relevante para identificar a dispersão das médias, onde um valor de dispersão baixo significa que a média está próxima da realidade.

Tabela 2. Médias de identificações de vulnerabilidades por etapas e grupos

	Inexperientes	Experientes
Etapa 1	0.25	1.25
Etapa 2	1	2
Etapa 3	2.75	3

Fonte: Elaborada pelo Autor.

5. Conclusão

Ao avaliar os resultados do experimento, foi possível concluir que a aplicação da metodologia CBR em atividades de identificação de vulnerabilidades é uma alternativa viável. Desta forma, a proposta estabelecida neste trabalho alcançou o seu objetivo de aproximar os resultados entre profissionais experientes e inexperientes em atividades de identificação de vulnerabilidades. Os objetivos de eliminação da dependência por desenvolvedores para atualização da ferramenta, assim como a possibilidade de detectar pequenas variações em uma definição de vulnerabilidade, também foram alcançados através da aplicação da metodologia CBR, onde a descrição de casos é realizada através de mecanismos amigáveis ao usuário final e a utilização de similaridade permite flexibilidade na detecção das vulnerabilidades.

6. Trabalhos Futuros

A aplicação da metodologia CBR para a identificação de vulnerabilidades oferece uma alternativa em relação aos sistemas baseados em regras tradicionais, entretanto é possível aperfeiçoá-lo ao agregar maior inteligência ao seu processo de aprendizado. Ao estruturar o protótipo deste trabalho, foi possível notar a oportunidade de agregar maior inteligência na recuperação de casos através da combinação de aprendizagem por reforço [32, p. 598] como meio de ordenar as melhores soluções entre os casos recuperados. Através da aplicação de aprendizagem por reforço, pode-se ordenar os casos de acordo com a frequência de utilização, como por exemplo, tornar obsoletos os casos pouco utilizados, permitindo então seu descarte e adicionando eficiência no algoritmo de recuperação de casos.

O mecanismo de recuperação de casos é dependente do sistema de pesos e desta forma, a atribuição de pesos se torna uma tarefa sensível e que requer conhecimento do contexto do caso. Identificou-se durante o desenvolvimento deste trabalho, a necessidade de um estudo para estabelecer as melhores métricas relacionadas aos pesos dos atributos, assim como a definição do melhor perfil de usuário para a atribuição destes valores em um ambiente organizacional. Durante a implementação da metodologia, identificou-se também questões relacionadas à eficiência e desempenho da estrutura de armazenamento de casos. Desta forma, análises de complexidade e custo de software podem ser aplicadas para fornecer um comparativo entre cada modelo estrutural, para subsidiar a escolha pela melhor opção.

Contribuições dos Autores

Os autores contribuíram igualmente na realização deste trabalho.

References

[1] GNU/LINUX Operational System. 2014. Disponível em: ¡http://www.gnu.org¿. Acesso em: 12 de abril de 2014.

[2] FOUNDATION, T. L. Linux adoption trends 2012: A survey of enterprise end users: A report by the linux foundation in partnership with yeoman technology group. 2012.

[3] OSBORN, S. Mandatory access control and role-based access control revisited. In: KIDWELLY, P. (Ed.). *IN PROCEEDINGS OF THE 2ND ACM WORKSHOP ON ROLE-BASED ACCESS CONTROL*. [S.l.]: ACM Press, 1997. p. 31–40.

[4] MüLLER, T. *ASLR Smack & Laugh Reference*. RWTH-Aachen University, 2008. (Technical report). Disponível em: ⟨http://www-users.rwth-aachen.de/Tilo.Mueller/ASLRpaper.pdf⟩.

[5] XU, H.; CHAPIN, S. J. Improving address space randomization with a dynamic offset randomization technique. In: *Proceedings of the 2006 ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2006. (SAC '06), p. 384–391. Disponível em: ⟨http://doi.acm.org/10.1145/1141277.1141364⟩.

[6] VEEN, V. Van der et al. Memory errors: The past, the present, and the future. In: *Proceedings of the 15th International Conference on Research in Attacks, Intrusions, and Defenses*. Berlin, Heidelberg: Springer-Verlag, 2012. (RAID'12), p. 86–106. Disponível em: ⟨http://dx.doi.org/10.1007/978-3-642-33338-5_5⟩.

[7] MITRE. 2014. Disponível em: ¡http://www.mitre.org¿. Acesso em: 12 de abril de 2014.

[8] NIST. *NIST - National Institute of Standards and Technology website*. [S.l.], 2014. Disponível em: ¡http://www.nist.gov¿. Acesso em: 12 de abril de 2014.

[9] FIRST. *FIRST Website*. [S.l.], 2014. Disponível em: ¡http://www.first.org¿. Acesso em: 12 de abril de 2014.

[10] WRIGHT, C. et al. Linux security modules: General security support for the linux kernel. In: *Proceedings of the 11th USENIX Security Symposium*. Berkeley, CA, USA: USENIX Association, 2002. p. 17–31. Disponível em: ⟨http://dl.acm.org/citation.cfm?id=647253.720287⟩.

[11] SCARFONE, K. A.; JANSEN, W.; TRACY, M. *SP 800-123. Guide to General Server Security*. Gaithersburg, MD,

United States: National Institute of Standards & Technology, 2008.

[12] SHARMA, A. et al. Ferret: A host vulnerability checking tool. In: *Proceedings of the 10th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'04)*. Washington, DC, USA: IEEE Computer Society, 2004. (PRDC '04), p. 389–394. Disponível em: ⟨http://dl.acm.org/citation.cfm?id=977407.978716⟩.

[13] BOELEN, M. *Lynis: Security and system auditing tool to harden Linux systems (and more)*. 2007. Disponível em: ¡https://cisofy.com/lynis/¿. Acesso em: 12 de abril de 2014.

[14] CIS, N. G. *TIGER: The Unix security audit and intrusion detection tool*. 2002. Disponível em: ¡http://www.nongnu.org/tiger/¿. Acesso em: 12 de abril de 2014.

[15] OPENVAS. *Open Vulnerability Assessment System*. [S.l.], 2014. Disponível em: ¡http://www.openvas.org¿. Acesso em: 12 de abril de 2014.

[16] WEBER, S.; KARGER, P. A.; PARADKAR, A. A software flaw taxonomy: Aiming tools at security. In: *Proceedings of the 2005 Workshop on Software Engineering for Secure Systems&Mdash;Building Trustworthy Applications*. New York, NY, USA: ACM, 2005. (SESS '05), p. 1–7. Disponível em: ⟨http://doi.acm.org/10.1145/1082983.1083209⟩.

[17] KRSUL, I. V. *Software Vulnerability Analysis*. Tese (Doutorado), West Lafayette, IN, USA, 1998. AAI9900214.

[18] MITRE. *MITRE CVE, Common Vulnerabilities and Exposures*. [S.l.], 2014. Disponível em: ¡http://cve.mitre.org/¿. Acesso em: 12 de abril de 2014.

[19] MAURO, D. R.; SCHMIDT, K. J. *Essential SNMP, Second Edition*. [S.l.]: O'Reilly Media, Inc., 2005.

[20] VEEN, V. Van der et al. Memory errors: The past, the present, and the future. In: *Proceedings of the 15th International Conference on Research in Attacks, Intrusions, and Defenses*. Berlin, Heidelberg: Springer-Verlag, 2012. (RAID'12), p. 86–106. Disponível em: ⟨http://dx.doi.org/10.1007/978-3-642-33338-5_5⟩.

[21] LANDWEHR, C. E. et al. A taxonomy of computer program security flaws. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 26, n. 3, p. 211–254, set. 1994. Disponível em: ⟨http://doi.acm.org/10.1145/185403.185412⟩.

[22] CVE-2014-0160. 2014. HeartBleed Vulnerability. Disponível em: ¡https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160¿. Acesso em: 12 de abril de 2014.

[23] PROVOS, N.; FRIEDL, M.; HONEYMAN, P. Preventing privilege escalation. In: *Proceedings of the 12th Conference on USENIX Security Symposium - Volume 12*. Berkeley, CA, USA: USENIX Association, 2003. (SSYM'03), p. 16–16. Disponível em: ⟨http://dl.acm.org/citation.cfm?id=1251353.1251369⟩.

[24] ABNT, A. B. d. N. T. *NBR ISO/IEC 27005 – Tecnologia da Informação – Técnicas de Segurança – Gestão de Riscos de Segurança da Informação*. Rio de Janeiro: ABNT, 2008.

[25] FRISCH, A. *Essential System Administration*. 3rd. ed. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2002. 387 p.

[26] MOURAD, A.; LAVERDIèRE, M.-A.; DEBBABI, M. Security hardening of open source software. In: *Proceedings of the 2006 International Conference on Privacy, Security and Trust: Bridge the Gap Between PST Technologies and Business Services*. New York, NY, USA: ACM, 2006. (PST '06), p. 43:1–43:1. Disponível em: ⟨http://doi.acm.org/10.1145/1501434.1501486⟩.

[27] NOONAN, W. *Hardening Network Infrastructure*. [S.l.]: McGraw-Hill Osborne Media, 2004.

[28] RED HAT INC. *Red Hat Enterprise Linux 6 Security Guide*. 5. ed. [S.l.], 2011. Disponível em: ⟨https://access.redhat.com/site/documentation/en-US/Red_Hat_Enterprise_Linux/6/pdf/Security_Guide/Red_Hat_Enterprise_Linux-6-Security_Guide-en-US.pdf⟩.

[29] WITA, R.; TENG-AMNUAY, Y. Vulnerability profile for linux. In: *Proceedings of the 19th International Conference on Advanced Information Networking and Applications - Volume 1*. Washington, DC, USA: IEEE Computer Society, 2005. (AINA '05), p. 953–958. Disponível em: ⟨http://dx.doi.org/10.1109/AINA.2005.343⟩.

[30] MILLS, D. et al. Rfc1157 - network time protocol version 4: Protocol and algorithms specification. 6 2010.

[31] LUGER, G. F. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. 3rd. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997.

[32] RUSSELL, S. J. et al. *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996.

[33] JACKSON, P. *Introduction to Expert Systems*. 2nd. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1990.

[34] RICHTER, M. M.; WEBER, R. O. *Case-Based Reasoning: A Textbook*. [S.l.]: Springer Publishing Company, Incorporated, 2013.

[35] PRICE, C. J.; PEGLER, I. S. Deciding parameter values with case-based reasoning. In: *Proceedings of the First United Kingdom Workshop on Progress in Case-Based Reasoning*. London, UK, UK: Springer-Verlag, 1995. p. 121–133. Disponível em: ⟨http://dl.acm.org/citation.cfm?id=646531.695375⟩.

[36] HOLT, A. et al. Medical applications in case-based reasoning. *Knowl. Eng. Rev.*, Cambridge University Press, New York, NY, USA, v. 20, n. 3, p. 289–292, set. 2005. Disponível em: ⟨http://dx.doi.org/10.1017/S0269888906000622⟩.

[37] VOSSOS, G. et al. An example of integrating legal case based reasoning with object-oriented rule-based systems: Ikbals ii. In: *Proceedings of the 3rd International Conference on Artificial Intelligence and Law*. New York, NY, USA: ACM, 1991. (ICAIL '91), p. 31–41. Disponível em: ⟨http://doi.acm.org/10.1145/112646.112650⟩.

[38] MELCHIORS CRISTINA; TAROUCO, L. M. R. *Raciocínio baseado em casos aplicado ao gerenciamento de falhas em redes de computadores*. UFRGS, 1999. Disponível em: ⟨http://www.lume.ufrgs.br/handle/10183/26310⟩.

[39] WATSON, I. Case-based reasoning is a methodology not a technology. In: MILES, R.; MOULTON, M.; BRAMER, M. (Ed.). *Research and Development in Expert Systems XV*. Springer London, 1999. p. 213–223. Disponível em: ⟨http://dx.doi.org/10.1007/978-1-4471-0835-1_15⟩.

[40] KOLODNER, J. *Case-based Reasoning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[41] KOLODNER, J. L.; SIMPSON, R. L.; SYCARA-CYRANSKI, K. A process model of cased-based reasoning in problem solving. In: *Proceedings of the 9th International Joint Conference on Artificial Intelligence - Volume 1*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1985. (IJCAI'85), p. 284–290. Disponível em: ⟨http://dl.acm.org/citation.cfm?id=1625135.1625188⟩.

[42] MAIN, J.; DILLON, T. S.; SHIU, S. C. K. Soft computing in case based reasoning. In: PAL, S. K.; DILLON, T. S.; YEUNG, D. S. (Ed.). London, UK, UK: Springer-Verlag, 2001. cap. A Tutorial on Case Based Reasoning, p. 1–28. Disponível em: ⟨http://dl.acm.org/citation.cfm?id=357410.357787⟩.

[43] GUPTA, K. M.; MONTEZEMI, A. R. Empirical evaluation of retrieval in case-based reasoning systems using modified cosine matching function. *Trans. Sys. Man Cyber. Part A*, IEEE Press, Piscataway, NJ, USA, v. 27, n. 5, p. 601–612, set. 1997. Disponível em: ⟨http://dx.doi.org/10.1109/3468.618259⟩.

[44] WATSON, I. Case-based reasoning is a methodology not a technology. In: MILES, R.; MOULTON, M.; BRAMER, M. (Ed.). *Research and Development in Expert Systems XV*. Springer London, 1999. p. 213–223. Disponível em: ⟨http://dx.doi.org/10.1007/978-1-4471-0835-1_15⟩.

[45] CUNNINGHAM, P. A taxonomy of similarity mechanisms for case-based reasoning. *IEEE Trans. on Knowl. and Data Eng.*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 21, n. 11, p. 1532–1543, nov. 2009. Disponível em: ⟨http://dx.doi.org/10.1109/TKDE.2008.227⟩.

[46] OSBORNE, H.; BRIDGE, D. Models of similarity for case-based reasoning. In: *Proceedings of the Interdisciplinary Workshop on Similarity and Categorisation*. [S.l.: s.n.], 1997. p. 173–179.

[47] SUEBSING, A.; HIRANSAKOLWONG, N. Feature selection using euclidean distance and cosine similarity for intrusion detection model. In: *Proceedings of the 2009 First Asian Conference on Intelligent Information and Database Systems*. Washington, DC, USA: IEEE Computer Society, 2009. (ACIIDS '09), p. 86–91. Disponível em: ⟨http://dx.doi.org/10.1109/ACIIDS.2009.23⟩.

[48] WATSON, I.; MARIR, F. Case-based reasoning: A review. *The Knowledge Engineering Review*, v. 9, p. 327–354, 12 1994. Disponível em: ⟨http://journals.cambridge.org/article_S0269888900007098⟩.

[49] LEWIS, L. *Managing Computer Networks: A Case-Based Reasoning Approach*. Norwood, MA, USA: Artech House, Inc., 1995.

[50] AAMODT, A.; PLAZA, E. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Commun.*, IOS Press, Amsterdam, The Netherlands, The Netherlands, v. 7, n. 1, p. 39–59, mar. 1994. Disponível em: ⟨http://dl.acm.org/citation.cfm?id=196108.196115⟩.

[51] BECHTSOUDIS, A.; SKLAVOS, N. Aiming at higher network security through extensive penetration tests. *Latin America Transactions, IEEE (Revista IEEE America Latina)*, v. 10, n. 3, p. 1752–1756, April 2012.

[52] PAVLOU, G. *On the Evolution of Management Approaches, Frameworks and Protocols: A Historical Perspective*. New York, NY, USA: Plenum Press, 2007. 425–445 p. Disponível em: ⟨http://dx.doi.org/10.1007/s10922-007-9082-9⟩.

[53] COCKBURN, A. *Writing Effective Use Cases*. 1st. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000.